Open in app ↗

Search

World's Simplest database | Using bash functions

Suresh Podeti
3 min read · May 13, 2023

Listen Share More

In this article I will explain how **database storage and retrieval works** with an example.

Why should you, as an application developer, care how the database handles storage and retrieval internally? You're probably not going to implement hour own storage engine from scratch, but you do need to select a storage engine that is appropriate for your application, from the many that are available. In order to tune a storage engine to perform well on your kind of workload, you need to have a rough idea of what the storage engine is doing under the hood.

Consider the **world's simplest key-value database**, implemented as **two Bash functions:**

```
#!/bin/bash
set() {
  echo "$1,$2" >> database
}

get(){
  grep "^$1," database | sed -e "s/^$1,//" | tail -n 1
}
```

Save above code in *db.sh* file and run *chmod u+x db.sh.* We can run this file as follows (Note: dot(.) in the command):

```
$ . ./db.sh; set <key> '<value>'
$ . ./db.sh; get <key>
```

These two functions **implement a key-value store**. You can call set key value which will store key and value in the database. The key and value can be (almost) anything you like — for example, the value could be a JSON document.

```
$ set 123456 '{"name": "Influence", "price": 234, "author": "Dr. Robert Cialdir
```

You can then call get key, which looks up the most recent value associated with that particular key and returns it.

```
$ get 123456
{"name": "Influence", "price": 234, "author": "Dr. Robert Cialdini"}
```

The underlying storage format is very simple: a **text file** where each line contains key-value pair, separated by a comma shown below(roughly like a csv file, ignoring escaping issues). Every call to set **append** to the end of the file, so if you update a key several times, the old versions of the value are **not overwritten** — you need to look at the last occurrence of a key in a file to find the latest value (hence we are using tail -n 1 in get).

```
$ cat database
123456,{"name": "Influence", "price": 234, "author": "Dr. Robert Cialdini"}
```

Let's say if we are updating the price of the book "influence" to Rs. 500.

```
$ set 123456 '{"name": "Influence", "price": 500, "author": "Dr. Robert Cialdir
```

It will append (not overwrite) as shown below:

```
$ cat database
123456,{"name": "Influence", "price": 234, "author": "Dr. Robert Cialdini"}
123456,{"name": "Influence", "price": 500, "author": "Dr. Robert Cialdini"}
```

Now, if we use get, will get updated value of the key.

```
$ get 123456
{"name": "Influence", "price": 500, "author": "Dr. Robert Cialdini"}
```

Our **set function** actually has **pretty good performance** for something that is so simple, because **appending to a file is generally very efficient.** Similarly to what set does, many database internally use a **log,** which is an **append-only** data file. Other than what is mentioned above real databases have more issues to deal with such as concurrency control, reclaiming disk space so that log does not grow forever, and handling errors and partially written records, but the basic principle is the same.

On the other hand, our **get function** has **terrible performance** if you have a large number of records in your database. Every time you want to look up a a key, get has to **scan the entire database from beginning to end,** looking for occurrences of the key. In algorithmic terms, the cost of a lookup is **O(*n*).** If you double the number of records n in your database, a lookup takes twice as long. That’s not good.

In order to efficiently find the value for a particular key in the database, we need a different data structure: an *index*. In my next article I will explain how ***indexing*** works in general.

References:

1. O'Reilly designing data-intensive applications by Martin Kleppmann, Chapter 3: Storage and Retrieval.

Database Key Value Database Indexing Storage Engine

Mysql Storage Engine

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience

Recommended from Medium